

# Technische Informatik 2 - Rechnerarchitektur

## Eigenes Skript

Studiengang Angewandte Informatik

Duale Hochschule Baden-Württemberg Karlsruhe

von

Andre Meyering

Kurs: TINF16B2  
Dozent: Prof. Dr. Jürgen Röthig  
Semester: 3. Semester (09.10.2017)  
*letzte Änderung:* 13. November 2017

Dies ist das eigene Skript für „Rechnerarchitektur“ bei Herrn Prof. Dr. Jürgen Röthig für das 3. Semester im Jahr 2017. Es enthält fast alles, was im Unterricht an die Tafel geschrieben oder besprochen wurde. Die  $\LaTeX$ -Dateien sollten sich im gleichen Share befinden, in dem du diese PDF-Datei gefunden hast.

Bei Fragen, Fehlern oder Ergänzungen – oder sollten die  $\LaTeX$ -Dateien fehlen – wende dich bitte an [dhbw@andremeyering.de](mailto:dhbw@andremeyering.de). Ich hoffe, diese PDF hilft dir beim Lernen.

## Inhaltsverzeichnis

<b>1</b>	<b>Vorwort</b>	<b>1</b>
<b>2</b>	<b>Einstieg in die Vorlesung</b>	<b>2</b>
2.1	Digitaltechnik (Vorlesung)	2
2.2	Übersicht	2
<b>3</b>	<b>Rechner</b>	<b>3</b>
3.1	Geschichte	3
3.2	Fundamentalarchitektur	4
<b>4</b>	<b>Rechenwerk / Rechnen in Hardware</b>	<b>8</b>
4.1	Schaltnetzsynthese – Wiederholung / Grundlegendes	8
4.2	Halbaddierer	9
4.3	Volladdierer	9
4.4	Paralleladdierer (4-Bit-Ripple-Carry-Paralleladdierer RC-PA)	10
4.5	Paralleladdierer (4-Bit-Carry-Look-Ahead-Paralleladdierer CLA-PA)	11
4.6	Serielladdierer	14
4.7	Subtraktion	16
4.8	Multiplikation	17
4.9	Division	21
4.10	CPUs	21
4.11	Speicher in Computern	22
<b>5</b>	<b>Abkürzungsverzeichnis</b>	<b>24</b>
	<b>Listingsverzeichnis</b>	<b>27</b>
	<b>Stichwortverzeichnis</b>	<b>28</b>

## 1 Vorwort

Herr Röthig schreibt alles, was für seine Klausuren von Bedeutung ist, an die Tafel. Es ist daher nur zu empfehlen, alles mitzuschreiben, da er kein Skript besitzt und auch keinen Foliensatz. Der Unterricht ist im Vergleich zu anderen Dozenten unterscheidet sich darin, dass während der Klausur keine Hilfsmittel verwendet werden dürfen. Dafür besteht die Klausur zu 90% nur aus Abfrageaufgaben.

Dieses Skript enthält *alles*, was Herr Röthig 2017 an Wissen voraussetzt. Auf den letzten Seiten dieses Skripts findet sich zusätzlich noch eine Übungsklausur. Die Klausuren unterscheiden sich jedes Jahr nur um einige wenige Aufgaben. Ist man zwei, drei Übungsklausuren durchgegangen, so ist die Klausur einfach zu bestehen.

Zusammen mit meinem Kurs TINF16B2 haben wir dieses Skript ausgedruckt und korrigiert. Inhaltliche Fehler sollten daher (fast) keine mehr enthalten sein.

Ich wünsche dir viel Erfolg bei Herrn Röthig im Fach Rechnerarchitektur (Technische Informatik 2). Solltest du diese Skript erweitern wollen, so kannst du dich an [dhbw@andremeyering.de](mailto:dhbw@andremeyering.de) wenden.

## 2 Einstieg in die Vorlesung

Dozent: Prof. Dr. Jürgen Röhlig  
Modul: Technische Informatik II  
Fach: Rechnerarchitektur

### 2.1 Digitaltechnik (Vorlesung)

- 36h/33h / 4h pro Woche
- Klausur: 21.12.2017 | 60min (ohne Hilfsmittel, Verrechnung mit Betriebssysteme)
- kein Skript, kein Foliensatz

### 2.2 Übersicht

1. Einführung, Begriffsbildung, Historie, Fundamentalarchitektur
2. Rechenwerke
3. Speicherwerk: Hauptspeicherorganisation  
Speicher: Cache, nicht-flüchtige Speichertechnologie
4. ausgewählte Kapitel aus Steuer-/Ein- und Ausgabewerke sind in 2 + 3 enthalten.

## 3 Rechner

### Rechner

Hilfsmittel zum Durchführen von „Rechnungen“.

- schneller
- fehlerfreier
- besseres Speichervermögen

### Rechenmaschine

- Abakus (mechanisch, digital)
- Rechenschieber (mechanisch, analog)

### Arbeitsweise

Man unterscheidet zwischen „mechanisch vs elektrisch“ und „digital vs analog“. Moderne „Rechner“ (PC & Co.) arbeiten elektrisch und digital. Dem gegenüber stehen elektrische Analogrechner, die um die 1920er genutzt wurden.

## 3.1 Geschichte

### 3.1.1 Elektrischer Digitalrechner

#### ZUSE Z1, Z2 (ab ~1940)

Relais als zentrale Bauteile (elektromagnetischer Schalter mit Elektromagnet)

- ⊕ Automatismus möglich
- ⊖ langsame Geschwindigkeit
- ⊖ großer Platzverbrauch
- ⊖ Geräusche beim Schalten
- ⊖ hoher Energieverbrauch beim Schalten
- ⊖ großer Verschleiß

#### ENIAC (~1945)

Die ENIAC besitzt als zentrales Bauteil eine Elektronenröhre. Eine Elektronenröhre ist ein eigentlich analog arbeitender Verstärker, wird hier aber als digitaler Schalter genutzt. Die Funktionsweise wird in Abbildung 3.1 dargestellt, wobei die Kathode negativ und die Anode positiv geladen sind.

- ⊕ sehr hohe Geschwindigkeit
- ⊖ großer Platzverbrauch
- ⊖ ständiges Summen bei 50Hz oft möglich
- ⊖ hoher, ständiger Energieverbrauch
- ⊖ großer Verschleiß

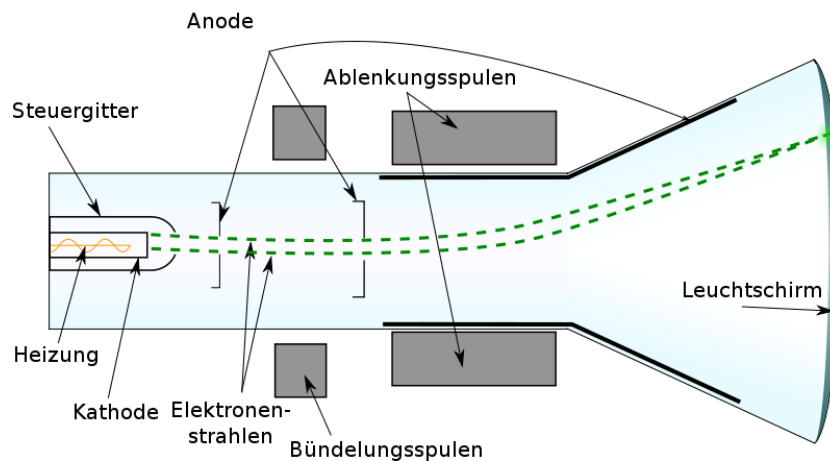


Abbildung 3.1: Funktionsweise Kathodenstrahlröhre [Quelle: Wikipedia]

**Moderne Rechner** Moderne transistorisierte Digitalrechner (z. B. UNIVAC ab Ende der 1950er).

- Transistor als zentrales Bauteil. Ein Transistor ist ein analog arbeitender Verstärker, wird hier aber als digital arbeitender Schalter genutzt.
- ⊕ sehr hohe Geschwindigkeit
- ⊕ sehr geringer Platzverbrauch
- ⊕ keine Geräuschentwicklung (außer Lüfter)
- ⊕ sehr niedriger Energieverbrauch
- ⊕ geringer Verschleiß

## 3.2 Fundamentalarchitektur

### 3.2.1 von-Neumann-Architektur

In Abbildung 3.2 wird die von-Neumann-Architektur vereinfacht dargestellt. Diese besteht aus:

**Zentraleinheit (CPU)** Die CPU besteht aus:

**Rechenwerk** Rechnen mit Zahlen und logischen Werten

**Steuerwerk** Zuständig für das Steuern und Koordinieren aller anderen Komponenten ⇒ Interpretation und Ausführung des (Maschinensprachen-)Programms

**Speicherwerk** (Hauptspeicher, Primärspeicher)

Speichern von Informationen (sowohl Programmcode als auch Nutzdaten gleichermaßen)

**Bus** verbindet alle Komponenten und ermöglicht den Informationsaustausch/Datenfluss zwischen ihnen.

**Eingabewerk** „Schnittstelle“ für Eingabegeräte (z. B. USB-Controller, S-ATA-Controller) nicht jedoch das Peripheriegerät selbst (also nicht die Tastatur)

**Ausgabewerk** „Schnittstelle“ für Ausgabegeräte (z. B. Grafikkarte)

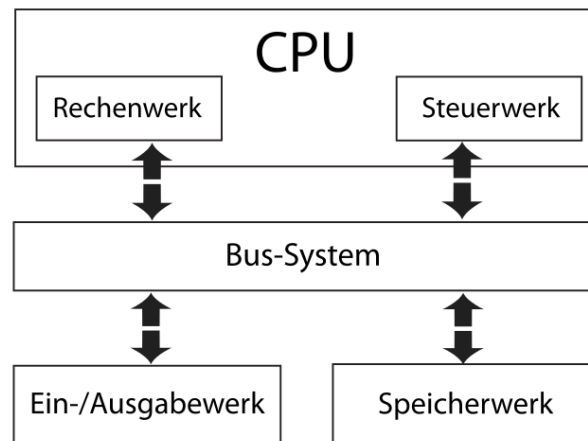


Abbildung 3.2: Vereinfachte Darstellung der von-Neumann-Architektur

### 3.2.2 Harvard-Architektur

Die Harvard-Architektur ist ähnlich der von-Neumann-Architektur, besitzt aber anstatt eines gemeinsamen, zwei getrennte Speicherwerke für Nutzdaten und Programmcode. Zusätzlich kann noch ein optionales zweites Eingabewerk existieren, welches nur für den Programmcode vorhanden ist. Das Speicher- und Eingabewerk für den Programmcode wird über einen zweiten Bus angebunden.

Dadurch ist eine klare physikalische Trennung von Programmcode und Nutzdaten möglich. Abbildung 3.3 auf der nächsten Seite zeigt die Harvard-Architektur und wie sich diese von der von-Neumann-Architektur unterscheidet.

### 3.2.3 Vergleich

Tabelle 3.1 auf Seite 7 vergleicht die von-Neumann-Architektur mit der Harvard-Architektur.

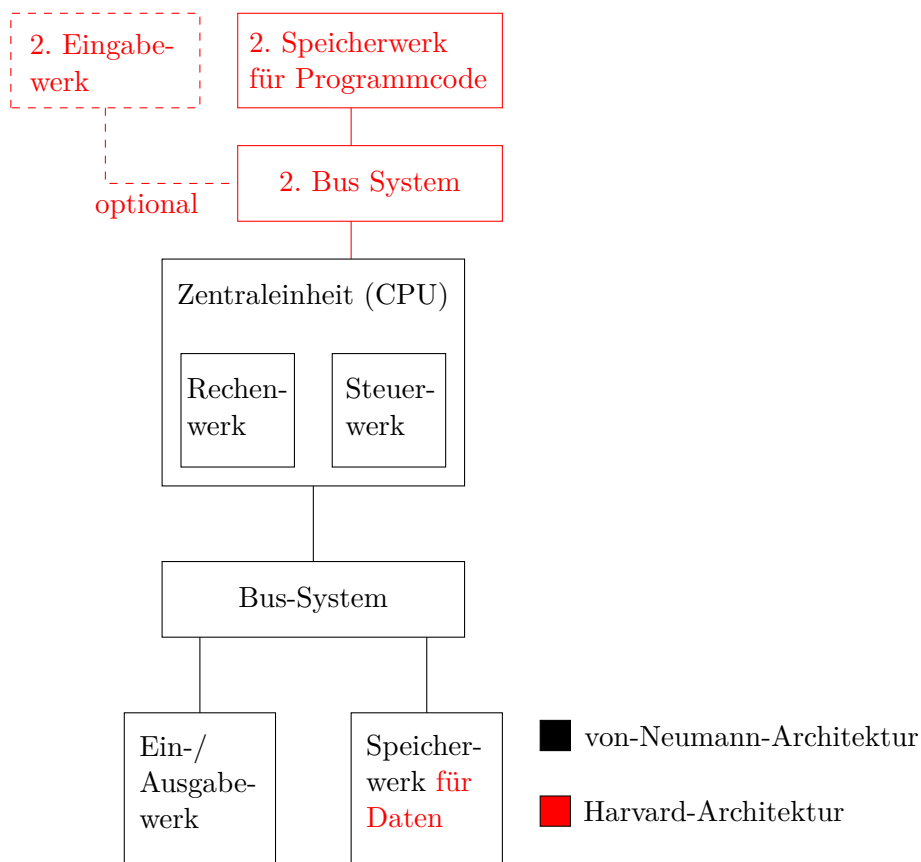


Abbildung 3.3: Vereinfachte Darstellung der Harvard-Architektur



von-Neumann-Architektur	Harvard-Architektur
<ul style="list-style-type: none"> <li>⊖ Virenanfälligkeit: Nutzdaten können als Programm ausgeführt werden</li> <li>⊕ universelle Programmierbarkeit (z. B. Compiler-Ausgabe wird als Programme ausgeführt)</li> <li>⊕ flexible Speicheraufteilung zwischen Programmcode und Daten</li> <li>⊖ möglicher Flaschenhals Bus &amp; Speicherwerk</li> <li>⊕ kostengünstig</li> </ul>	<ul style="list-style-type: none"> <li>⊕ nahezu immun gegen unabsichtlichen Virenbefall</li> <li>⊕ keine (unbeabsichtigten oder ungewollten) Änderungen an der Betriebssoftware möglich</li> <li>⊖ komplexer und teurer (durch 2 Bus und ggf. 2 Eingabewerke)</li> <li>⊖ schwer update-fähig</li> <li>⊕ bessere Performance möglich durch gleichzeitigen Zugriff auf beide Speicherwerke</li> <li>⊖ ohne zweites Eingabewerk keine Möglichkeit anderen Programmcode auszuführen.</li> <li>⊖ unflexible Aufteilung des Speichers, wenn das eine Speicherwerk voll ist, kann das andere Speicherwerk nicht genutzt werden.</li> </ul>
Einsatz	
<ul style="list-style-type: none"> <li>• übliche PC-Architektur – „Universal-PC“</li> </ul>	<ul style="list-style-type: none"> <li>• „embedded systems“ (z. B. in Waschmaschinen, KFZ-Elektronik, etc.)</li> <li>• Smartphones &amp; Co.</li> <li>• Bestandteile von PCs: BIOS, CPU-Cache in modernen CPUs (Trennung in Cache für Programmcode und Nutzdaten), NX-Flag (Non-Executable) im Hauptspeicher</li> </ul>

Tabelle 3.1: Vergleich der von-Neumann- und Harvard-Architektur

## 4 Rechenwerk / Rechnen in Hardware

### Grundlagen: Addition

	4 7 1 1	Die Addition von mehrstelligen Zahlen wird reduziert auf die Addition von zwei (oder drei) einstelligen Zahlen (bzw. Ziffern) zu einer einstelligen Zahl sowie einem einstelligen Übertrag, also einer zweistelligen Zahl als Ergebnis $\Rightarrow$ genauso funktioniert dies im Binärsystem.
	+ 0 8 1 5	
Übertrag	0 1 0 0	
	5 5 2 6	

#### Hinweis

Herr Röhlig meint mit „einstellig“ und „mehrstellig“ die Anzahl der Zahlen und nicht die Stellenanzahl einer einzelnen Zahl. Im Beispiel oben werden zwei Zahlen addiert und daraus ergeben sich zwei weitere Zahlen als Ergebnis.

### 4.1 Schaltnetzsynthese – Wiederholung / Grundlegendes

**Schaltnetz** Kann nur die derzeitigen Eingangsdaten verarbeiten, da keine Rückkopplung vorliegt

**Schaltwerk** „Hat ein Gedächtnis“, da eine Rückkopplung vorliegt

**Vollkonjunktion/Minterm** UND-Verknüpfung aller vorkommenden Variablen entweder in negierter oder nicht-negierter Form

**Disjunktive Normalform (DNF)** Eine Disjunktion (ODER-Verknüpfung) von Konjunktionstermen (UND-Verknüpfungen).

**Disjunktive Minimalform (DMF)** Ist die minimale Darstellung einer Ausgabefunktion und damit eine Vereinfachung einer DNF

#### 4.1.1 Schaltungsanalyse

Eine Schaltungsanalyse ist die Bestimmung des „Aufwands“. Dabei kann der Aufwand sein:

- „Hardware-Aufwand“ (in Anzahl an Transistoren)
- Zeitaufwand (in Gatterlaufzeit)

#### 4.1.2 Warum soll der Zeitaufwand analysiert werden?

Es wird der Zeitaufwand betrachtet, da Gatter Schaltzeiten haben, welche typischerweise  $\sim 10$  Pikosekunden betragen. Insgesamt werden bei einem Signaldurchgang auf dem IC sehr viele Gatter durchlaufen. Damit sind die Schaltzeiten um Größenordnungen größer als die reine Laufzeit der Signale auf dem Leiter angegeben (letztere wird vernachlässigt, Zeitverzögerung wird in „Anzahl Gatterlaufzeiten (GLZs)“ angegeben).

### 4.2 Halbaddierer

Vollzieht die Addition von zwei einstelligen Binärzahlen  $a$  und  $b$  zu einer zweistelligen Binärzahl  $c_{out}s$  (Übertrag und Summe). Schaltsymbol und Schaltnetz des Halbaddierers werden in Abbildung 4.1 dargestellt.

Die folgende Tabelle zeigt den Gedankenweg, wie ein Halbaddierer funktioniert.

Nr.	$b$	$a$	$c_{out}$	$s$	Minterm	DNF
0	0	0	0	0		$c_{out} = ba$ $s = \bar{b}a \vee b\bar{a}$ $\Rightarrow$ beides gleichzeitig auch DMF
1	0	1	0	1	$\bar{b}a$	
2	1	0	0	1	$b\bar{a}$	
3	1	1	1	0	$ba$	

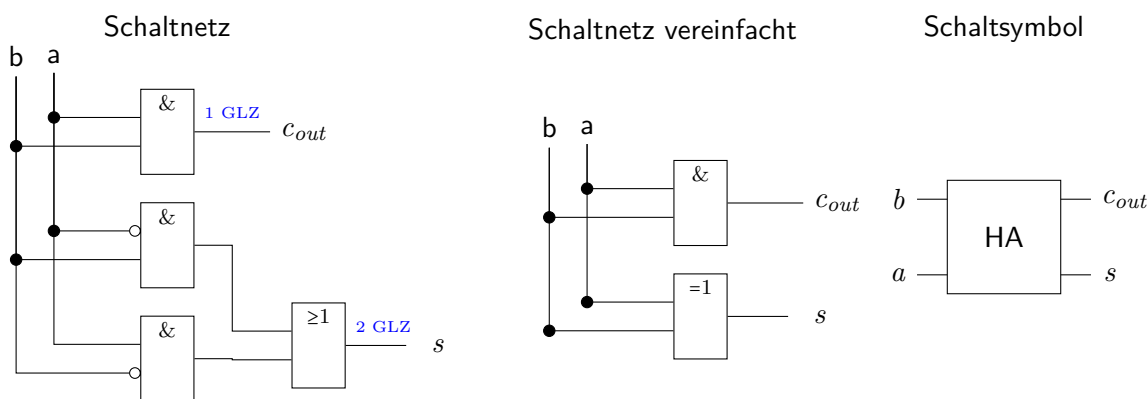


Abbildung 4.1: Halbaddierer – Schaltnetz und Schaltsymbol

### 4.3 Volladdierer

Vollzieht die Addition von drei einstelligen Binärzahlen  $a$ ,  $b$  und  $c_{in}$  zu einer zweistelligen Binärzahl  $c_{out}s$  (Übertrag und Summe). Schaltsymbol und Schaltnetz des Volladdierers werden in Abbildung 4.2 dargestellt.

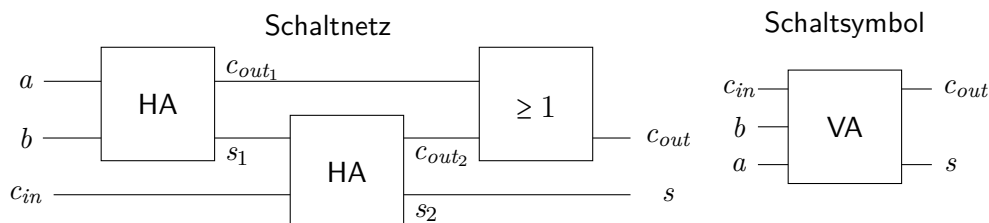


Abbildung 4.2: Volladdierer – Schaltnetz und Schaltsymbol

*Hinweis:* Für die Verknüpfung von  $c_{out_1}$  und  $c_{out_2}$  zu  $c_{out}$  wäre eigentlich ein XOR notwendig. Da aber der Fall  $c_{out_1} = c_{out_2} = 1$  (also beide Eingänge des XOR „1“) nie auftritt, reicht ein OR.

#### 4.4 Paralleladdierer (4-Bit-Ripple-Carry-Paralleladdierer RC-PA)

Der RC-PA ist ein mehrstelliger Addierer für Binärzahlen. In den folgenden Beispielen ist er ein Addierer vierstelliger Binärzahlen  $a_3 a_2 a_1 a_0$  und  $b_3 b_2 b_1 b_0$ . Das Ergebnis ist  $s_4 s_3 s_2 s_1 s_0$  und somit eine 5-stellige Zahl.  $s_4$  ist der Überlauf.

Abbildung 4.3 zeigt das Schaltnetz und Schaltsymbol eines Paralleladdierers.

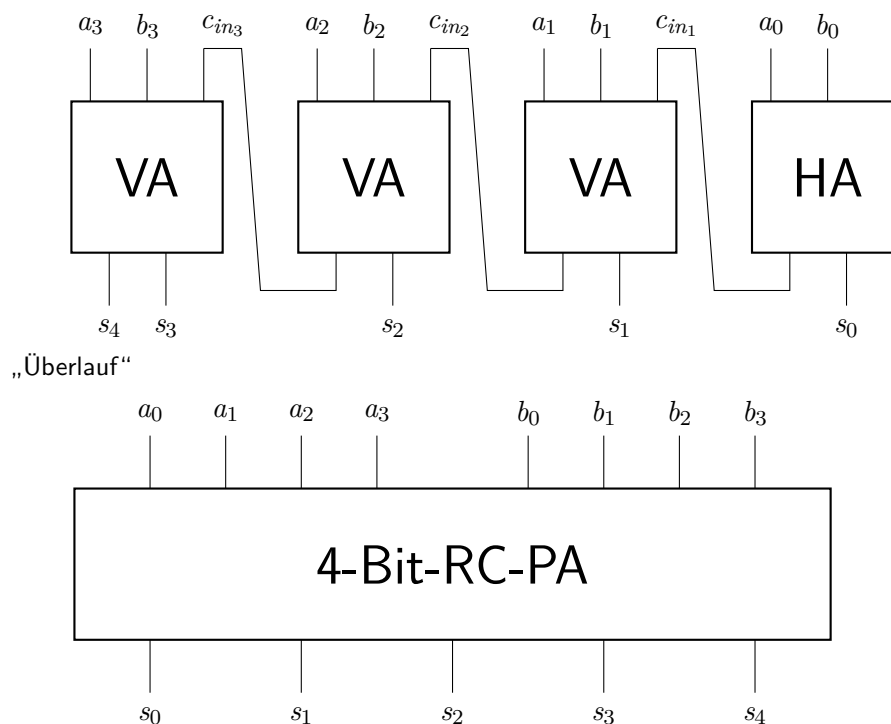


Abbildung 4.3: RC-Paralleladdierer – Schaltnetz und Schaltsymbol

#### Hinweis

Ein  $n$ -Bit RC-PA ist ein Schaltnetz, kein Schaltwerk! Eine zeichnerische Anordnung mit Verbindungen nur nach unten ist nämlich möglich.

#### 4.4.1 Hardwareaufwand des 4 Bit RC-PA

für HA: 2 Transistoren für  $c_{out}$  und 6 Transistoren für  $s$   $\Rightarrow$  8 Transistoren

für VA: 2 HA und 2 Transistoren für  $c_{out}$   $\Rightarrow$   $2 \cdot 8 + 2 = 18$  Transistoren

Für den 4 Bit RC-PA werden benötigt:

$$\begin{aligned}
 1 \text{ HA} + (n-1) \text{ VA} &\Rightarrow 8 \text{ Transistoren} + (n-1) \cdot 18 \text{ Transistoren} \\
 &\Rightarrow 8 + (18n - 18) \text{ Transistoren} = 18n - 10 \text{ Transistoren} \\
 &= O(n)
 \end{aligned}$$

Dies heißt, dass der HW-Aufwand linear mit der Breite der Summanden steigt. Dies ist gut, denn besseres (also weniger Aufwand) ist kaum zu erwarten.

#### 4.4.2 Zeitaufwand des 4 Bit RC-PA

für HA: max. 2 Gatterlaufzeiten (GLZs) („Tiefe 2“, siehe Abbildung 4.1 auf Seite 9)

für VA: max. 4 Gatterlaufzeiten (GLZs)

Beim RC-PA liegen die einzelnen  $s_i$  nach unterschiedlicher Zeit an.  $s_i$  wird nach  $(i + 1) \cdot 2$  GLZ erreicht. Das längste  $s_i$  ist bei  $n$ -Bit-RC-PA  $i = n - 1$  und damit ergibt sich ein Zeitaufwand bei  $n$ -Bit-RC-PA von  $2n$  GLZ!

Dies ist ein schlechter Zeitaufwand bei einem Paralleladdierer, denn zu erwarten wäre  $O(1)$ !

*Auswirkung:* Beim Wechsel von 32- auf 64-Bit-CPU hätte sich die Taktfrequenz halbiert. Daraus lässt sich folgern, dass kein 64-Bit-RC-PA in der CPU verbaut ist.

#### Hinweis

Anmerkung zum RC-PA: Die  $2n$  GLZ Zeitaufwand werden nur im „schlimmsten Fall“ bei einer ununterbrochene Kette von  $c_{out}$ -Ausgängen, welche sich **alle** im Laufe der Berechnung von 0 auf 1 ändern, erreicht.

Diese Zeit muss aber trotzdem abgewartet werden.

#### 4.5 Paralleladdierer (4-Bit-Carry-Look-Ahead-Paralleladdierer CLA-PA)

*Idee:* Der  $c_{in}$ -Eingang wird nicht von vorausgehenden VA (oder HA) übernommen, sondern durch ein „magisches CLA-Schaltnetz“ nachberechnet. *Genauer:* Für die Berechnung von  $c_{in_i}$  müssen alle vorherigen Eingänge  $a_j, b_j, j < i$  berücksichtigt werden. Abbildung 4.4 zeigt dieses „magische CLA-Schaltnetz“.

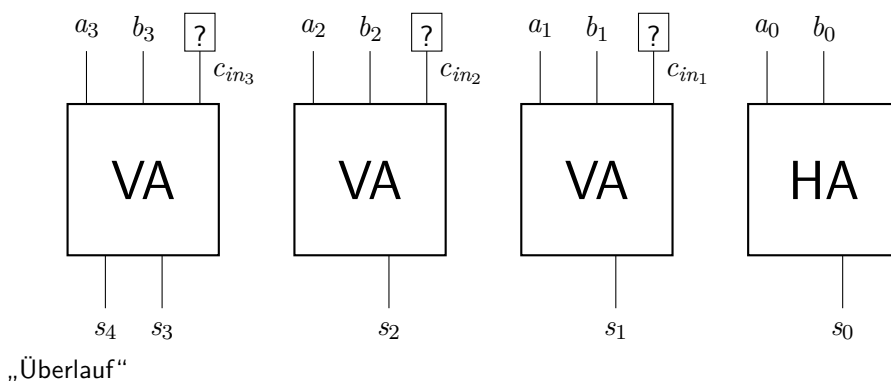


Abbildung 4.4: Carry-Look-Ahead-Paralleladdierer – Schaltnetz

Aber wie sieht das Schaltnetz (und die boolesche Formel) für die Berechnung von  $c_{in_i}$  aus?

$$\text{HA : } c_{out} = a \wedge b$$

$$s_{out} = a\bar{b} \vee \bar{a}b$$

$$\text{VA : } c_{out} = (a \wedge b) \vee (c_{in} \wedge (a\bar{b} \vee \bar{a}b))$$

$$c_{in_1} = c_{out_0} = a_0 \wedge b_0$$

HA

$$c_{in_2} = c_{out_1} = (a_1 \wedge b_1) \vee (c_{in_1} \wedge (a_1\bar{b}_1 \vee \bar{a}_1b_1))$$

VA

$$= (a_0 \wedge b_0) \vee (a_0 \wedge b_0 \wedge (a_1\bar{b}_1 \vee \bar{a}_1b_1))$$

Einsetzen

$$c_{in_3} = c_{out_2} = (a_2 \wedge b_2) \vee (c_{in_2} \wedge (a_2\bar{b}_2 \vee \bar{a}_2b_2))$$

VA

$$= (a_2 \wedge b_2) \vee ((a_0 \wedge b_0) \vee (a_0 \wedge b_0 \wedge (a_1\bar{b}_1 \vee \bar{a}_1b_1))) \wedge$$

Einsetzen

$$(a_2\bar{b}_2 \vee \bar{a}_2b_2)$$

...

#### 4.5.1 Zeitaufwand

Für die Berechnung der  $c_{in_i}$  kann jeweils eine DNF, DMF oder jede andere DxF (oder KxF) verwendet werden. Diese haben jeweils nur genau (bzw. maximal) 2 GLZ Zeitaufwand.

Insgesamt hat jedes  $s_i$  beim CLA-PA genau 6 GLZ Zeitaufwand (außer  $s_0$ : 2 GLZ,  $s_1$ : 5 GLZ). Somit haben wir einen konstanten Zeitaufwand von  $O(1)$ .

#### 4.5.2 Hardwareaufwand des 4 Bit CLA-PA

Für den Aufwand für die  $c_{in_i}$ -Berechnung gilt die Annahme, das Schaltnetz wäre eine Realisierung der DNF.

⇒ Für jedes  $c_{in_i}$  gibt es insgesamt  $2i$  Eingänge.

⇒ Insgesamt gibt es max.  $2^{2i}$  verschiedene Vollkonjunktionen, welche in der DNF auftreten können.

Jede dieser Vollkonjunktionen wird mit  $2i$  Transistoren realisiert.

⇒ Falls alle Vollkonjunktionen verwendet werden müssten, wäre der Hardwareaufwand  $2^{2i} \cdot 2i$  Transistoren.

Evtl. nicht korrekt. Die 15er haben da was anderes...

In der Realität werden natürlich nicht alle Vollkonjunktionen benötigt, sondern ein (vermutlich halbwegs konstanter) Anteil  $0 < k < i$ . Damit ist der Aufwand für  $c_{in_i} = O(i \cdot 4^i)$  und somit der Aufwand für  $n$ -Bit-CLA-PA:  $O(n^2 \cdot 2^n)$

**Achtung**

Im folgenden wird fälschlicherweise von einem Aufwand  $O(n^2 \cdot 2^n)$  ausgegangen. Richtig wäre  $O(n^2 \cdot 4^n)$ . Herr Röthig hat die  $O$ -Notation falsch vereinfacht. Der 2015er Jahrgang hat dies „noch falscher“ gemacht und zu  $O(4^n)$  vereinfacht.

Der Hardwareaufwand steigt beim  $n$ -Bit-CLA-PA überexponentiell mit  $n$ . Beim Wechsel von 32-Bit auf 64-Bit-CLA-PA wäre der 16 Trillionen-fache Aufwand an Transistoren nötig gewesen.

Bei  $n = 4$ :  $4^2 \cdot 2^4 = 16 \cdot 16 = 256$  Transistoren

Bei  $n = 8$ :  $8^2 \cdot 2^8 = 64 \cdot 256 = \sim 16384$  Transistoren (64-fache von  $n = 4$ )  $\Rightarrow$  zu viel für die CPU

Vergleichen:  
Siehe Notizen,  
etc.

**4.5.3 Kombination mehrerer kleinen CLA-PAs**

Der 32-Bit Addierer wird in acht 4-Bit-CLA-PA gesplittet (siehe Abbildung 4.5).

$\Rightarrow$  hintereinander geschaltet nach RC-Prinzip

$\Rightarrow$  damit ist das  $n$  der nicht-CLA-PA noch klein  $\Rightarrow$  erträglicher Hardwareaufwand

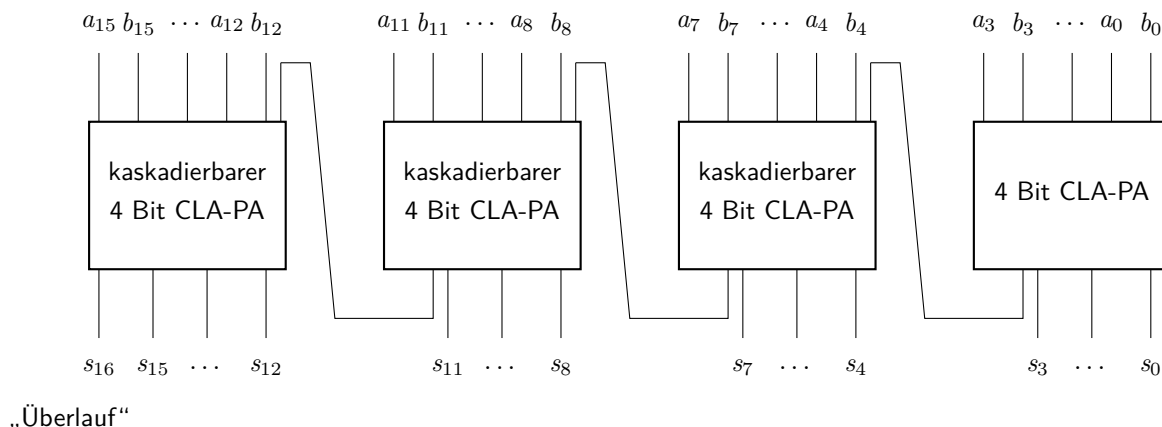


Abbildung 4.5: Kaskadierbarer 4-Bit-CLA-PA

## 4.6 Serielladdierer

*Idee:* Angelehnt an die Verfahrensweise des Menschen sollen die Stellen der beiden Summanden nacheinander (und nicht gleichzeitig) addiert werden. Dadurch wird nur **ein** VA und mehrere Schieberegister (SR) benötigt. Daher ist der Serielladdierer (SA) ein Schaltwerk, kein Schaltnetz! Abbildung 4.6 zeigt das Schaltwerk eines Serielladdierer.

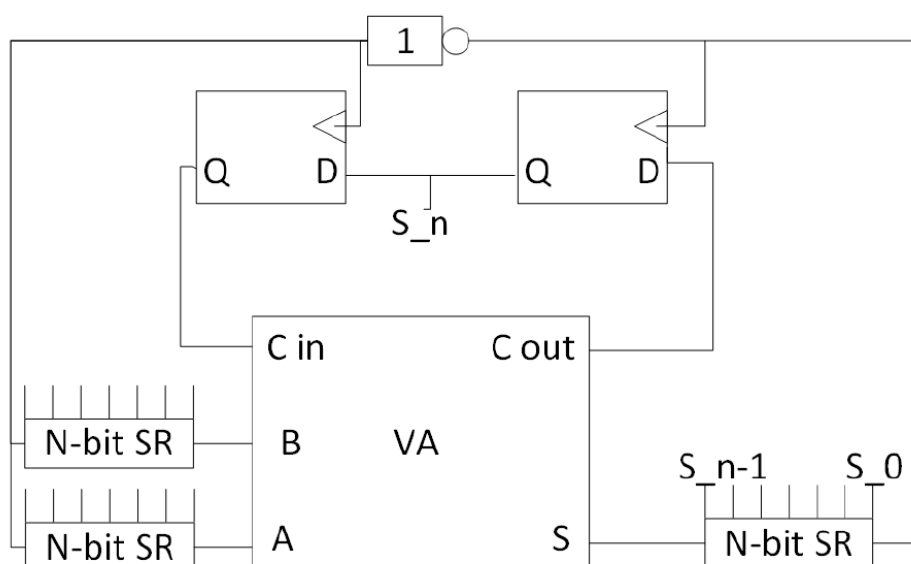


Abbildung 4.6: Serielladdierer

Als Tikz übernehmen

### 4.6.1 Zeitaufwand ( $n$ -Bit-SA)

Der Zeitaufwand für einen  $n$ -Bit-SA beträgt  $n$  Taktzyklen, also  $O(n)$

*Ist dies wie beim RC-PA?*

Jein, denn 1 Taktzyklus dauert deutlich mehr als doppelt so lang wie die Berechnung des VA (Sicherheitsmargen!). Beispiel: 1 Taktzyklus  $>$   $\sim 10$  GLZ  $\Rightarrow$  fünffache Berechnungszeit des RC-PA

### 4.6.2 Hardwareaufwand ( $N$ -Bit-SA)

1 VA = 18 Transistoren

2 D-FF =  $2 \cdot 6 = 12$  Transistoren. (siehe Abbildung 4.7 rechts)

3  $n$ -Bit-SR =  $4 \cdot 6n = 18n$  Transistoren (siehe Abbildung 4.7 links)

Takterzeugung (im folgenden nicht näher betrachtet)

gesamt  $18n + 30$  Transistoren

Zum Vergleich: RC-PA:  $18n - 10$ , d. h. der SA braucht 40 Transistoren mehr (bei längerer Bearbeitungszeit)!



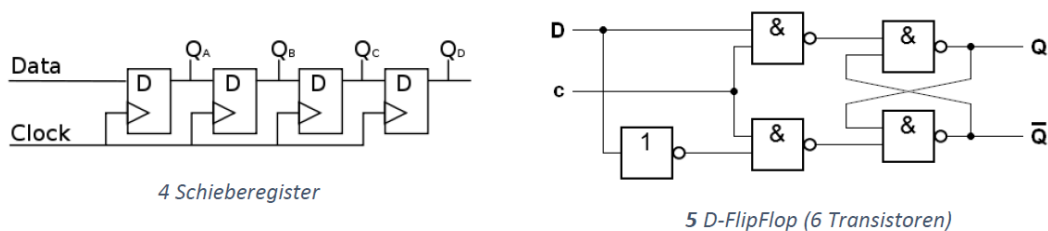


Abbildung 4.7: Schieberegister und D-Flip-Flop

**Achtung**

Die Takterzeugung muss in der Klausur für den Serielladdierer auf jeden Fall genannt werden, auch wenn sie hier nicht weiter betrachtet wird!

**4.6.3 Hardwareoptimierung des Serielladdierers**

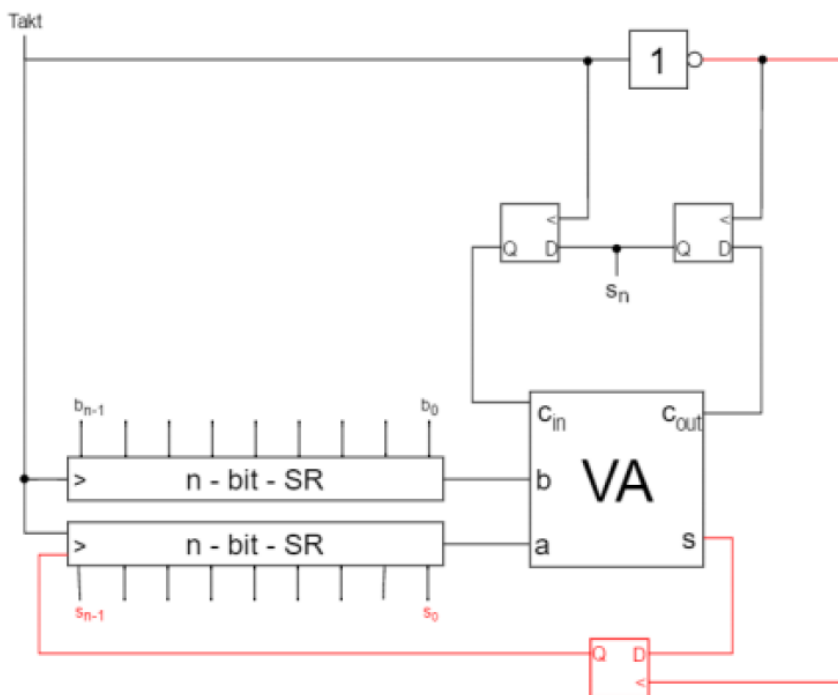


Abbildung 4.8: Serielladdierer mit Hardwareoptimierung

Wie in Abbildung 4.8 zu sehen ist, wird ein Schieberegister weniger benötigt  $\Rightarrow 12n + 36$  Transistoren

**Vergleich eines RC-PA mit dem verbesserten SA**

Tabelle Tabelle 4.1 vergleicht einen RC-PA mit dem verbesserten SA.

n	RC-PA (18n - 10)	SA (12n + 36)
1	8	48
2	26	60
3	44	72
4	62	84

Tabelle 4.1: Vergleich von Hardwareaufwand eines RC-PA mit dem verbesserten SA

Break-Even („Gewinnschwelle“):  $18n - 10 = 12n + 36 \Rightarrow 46 = 6n \Rightarrow n = 7\frac{2}{3}$   
 $\Rightarrow$  Ab 8-Bit lohnt sich der SA

**Mögliche Anwendung eines Serielladdierer:**

- nicht im Rechenwerk der CPU (aufgrund der Geschwindigkeit)!
  - aber möglicherweise in „Embedded System“, falls z. B. Sensordaten sowieso seriell angeliefert werden
- $\Rightarrow$  eventuell sind sogar weitere Hardware-Einsparungen möglich!

**4.7 Subtraktion**

Es ist kein spezieller Hardware-Subtrahierer notwendig. Eine Subtraktion wird über die Addition des 2er-Komplements realisiert:

1. Bits invertieren (Inverter)
2. +1 addieren (Addierer)

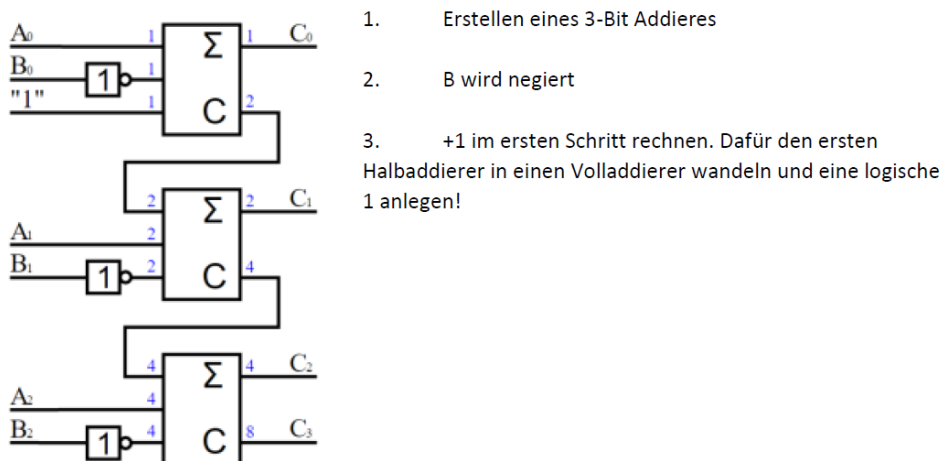


Abbildung 4.9: Schaltnetz Subtrahierer

### 4.8 Multiplikation

$  \begin{array}{r}  4711 \times 815 \\  \hline  35 \\  67688 \\  4711 \\  23 \\  23555 \\  \hline  3839465  \end{array}  $	<p><b>Mögliche Probleme / Schwierigkeiten</b></p> <ol style="list-style-type: none"> <li>1. kleines <math>1 \times 1</math> ist ein wenig Lernaufwand</li> <li>2. Überträge beim kleinen <math>1 \times 1</math> sind möglich</li> <li>3. Addition von mehr als zwei Ziffern gleichzeitig</li> <li>4. mehrstellige Überträge bei der Summenbildung möglich</li> <li>5. für jede Zwischensumme muss der Addierer eine Stelle mehr verarbeiten</li> </ol>
---	---

Punkt 5 wird deutlich durch eine Multiplikation einer 6- und 4-stelligen Zahl im Binärsystem „von links nach rechts“, wie sie in Tabelle 4.2 dargestellt wird.

$101010 \times 1010 \quad (42 \times 10)$
$  \begin{array}{r}  101010 \\  000000 \quad 7\text{-stellige Addition} \\  \hline  1010100 \quad \text{Zwischensumme} \\  101010 \quad 8\text{-stellige Addition} \\  \hline  11010010 \quad \text{Zwischensumme} \\  000000 \quad 9\text{-stellige Addition} \\  \hline  110100100 \quad 10\text{-stelliges Ergebnis}  \end{array}  $

Tabelle 4.2: Schriftliche Multiplikation „von links nach rechts“

#### Abhilfe

<table border="1" style="border-collapse: collapse; text-align: center;"> <tr> <td style="padding: 2px;">b</td> <td style="padding: 2px;">a</td> <td style="padding: 2px;">ab</td> </tr> <tr> <td style="padding: 2px;">0</td> <td style="padding: 2px;">0</td> <td style="padding: 2px;">0</td> </tr> <tr> <td style="padding: 2px;">0</td> <td style="padding: 2px;">1</td> <td style="padding: 2px;">0</td> </tr> <tr> <td style="padding: 2px;">1</td> <td style="padding: 2px;">0</td> <td style="padding: 2px;">0</td> </tr> <tr> <td style="padding: 2px;">1</td> <td style="padding: 2px;">1</td> <td style="padding: 2px;">1</td> </tr> </table>	b	a	ab	0	0	0	0	1	0	1	0	0	1	1	1	<ol style="list-style-type: none"> <li>1. Kleines <math>1 \times 1</math> im Binärsystem ist ein einfaches UND</li> <li>2. Es gibt keine Überträge beim kleinen <math>1 \times 1</math> im Binärsystem</li> <li>3. Bilden von Zwischensummen</li> <li>4. keine mehrstelligen Überträge bei Addition von zwei Summanden</li> <li>5. rechten Faktor beginnend mit niederwertigster Stelle abarbeiten, siehe Tabelle 4.3</li> </ol>
b	a	ab														
0	0	0														
0	1	0														
1	0	0														
1	1	1														

$101010 \times 1010 \quad (42 \times 10)$
$  \begin{array}{r}  00000\mathbf{0} \\  101010\mathbf{p_0} \\  \hline  10101\mathbf{0} \\  000000\mathbf{p_1} \\  \hline  01010\mathbf{1} \\  101010\mathbf{p_2} \\  \hline  0110100 \\  \mathbf{p_9 p_8 p_7 p_6 p_5 p_4 p_3}  \end{array}  $

Tabelle 4.3: Schriftliche Multiplikation „von rechts nach links“

### 4.8.1 Multiplikation mit Paralleladdierer

Ein  $n \times m$ -Bit-Parallelmultiplizierer (PM), bspw. ein  $5 \times 4$ -Bit-Parallelmultiplizierer ist in Abbildung 4.10 dargestellt.

Schaltnetz

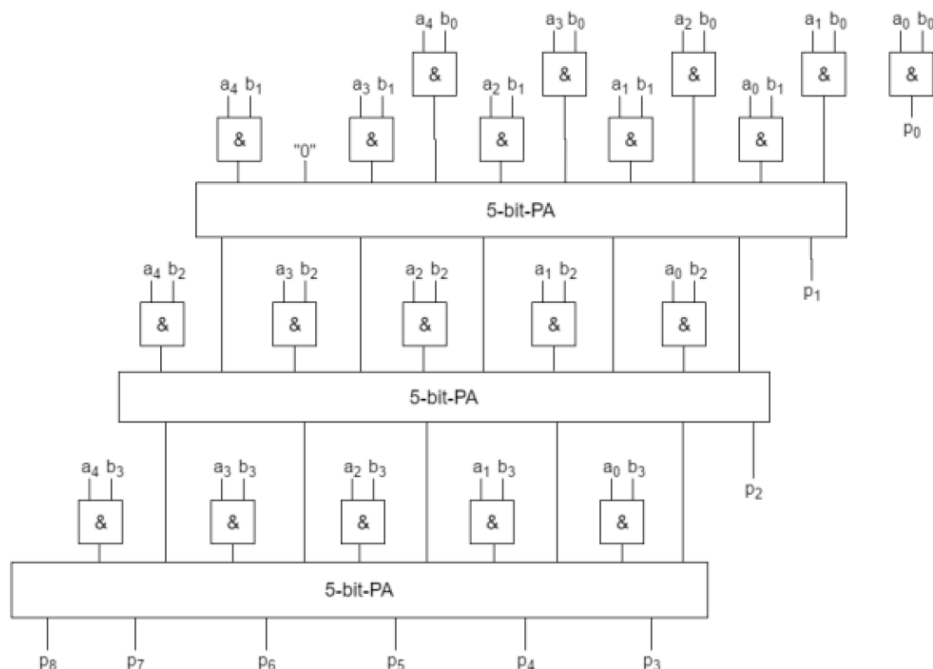


Abbildung 4.10: 5-Bit Parallelmultiplizierer

#### Hinweis zur Abbildung

Unbedingt auf die „0“ als Eingang achten! Ansonsten gibt es in der Klausur punktabzug!

#### Hinweis

In der Klausur muss evtl. ein 4-Bit Parallelmultiplizierer gezeichnet werden, also nicht mit 5 oder 6 Bit.

#### Analyse: Hardwareaufwand für die Multiplikation mit Paralleladdierer

Benötigt werden:

- $m - 1$   $n$ -Bit-PA
- $n \cdot m$  UND mit jeweils 2 Eingängen  $\Rightarrow 2 \cdot n \cdot m$  Transistoren

Somit ergibt sich bei...

... Verwendung eines **RC-PA**:

$\Rightarrow n$ -Bit-RC-PA:  $18n - 10$  Transistoren

$\Rightarrow$  davon  $m - 1$ :  $(m - 1)(18n - 10)$  Transistoren =  $18nm - 18n - 10m + 10$  Transistoren  
 $\Rightarrow$  Insgesamt:  $20nm - 18n - 10m + 10$  Transistoren  $\Rightarrow O(n)$

... Verwendung eines **CLA-PA**

- $n$ -Bit-CLA-PA:  $\approx O(n \cdot 2^n)$
- $\Rightarrow$  davon  $m - 1$ :  $(m - 1) \cdot O(n \cdot 2^n) = O(n \cdot m \cdot 2^n)$

Hier bei verschiedenen großen Faktoren also besser  $m > n$  bei Carry-Look-Ahead-Paralleladdierer (CLA-PA)) für geringeren HW-Aufwand.

Demgegenüber bei Verwendung von RC-PA: Besser  $n > m$  für geringeren HW-Aufwand.

Besser umformulieren

### Analyse: Zeitaufwand für die Multiplikation mit Paralleladdierer

1 GLZ für einstellige Multiplikation (UND-Gatter) sowie  $(m - 1) \times$  Berechnungszeit( $n$ -Bit-PA)

1. Annahme: PA sind  $n$ -Bit-RC-PA.

Berechnungszeit eines  $n$ -Bit-RC-PA:  $2n$  GLZ

Insgesamt:  $1 + (m - 1) \cdot 2n = 2nm - 2n + 1$  GLZ

Damit besser  $n > m$  bei Verwendung von RC-PA um geringeren Zeitaufwand zu bekommen.

2. Annahme: PA sind  $n$ -Bit-CLA-PA

Berechnungszeit eines  $n$ -Bit-CLA-PA:  $6$  GLZ

Insgesamt:  $1 + (m - 1) \cdot 6$  GLZ =  $6m - 5$  GLZ =  $O(m)$

damit: besser  $n > m$  bei Verwendung von CLA-PA, um geringeren Zeitaufwand zu bekommen

⚡ zu großer HW-Aufwand (wächst exponentiell mit  $n$ )

Kontrollieren...

#### Hinweis für die Klausur

Logarithmischen Aufwand für CLA-PA auf Wikipedia nachschauen. Dies wird wahrscheinlich in der Klausur abgefragt! Unserer Variante hat exponentiellen Hardwareaufwand und konstanten Zeitaufwand. Die Variante auf Wikipedia nicht.

### 4.8.2 Seriellmultiplizierer

Motivation: Noch engere Anlehnung an das schriftliche Multiplikationsverfahren, um den Aufwand für die Addierglieder gering zu halten.

Siehe Abbildung 4.11.

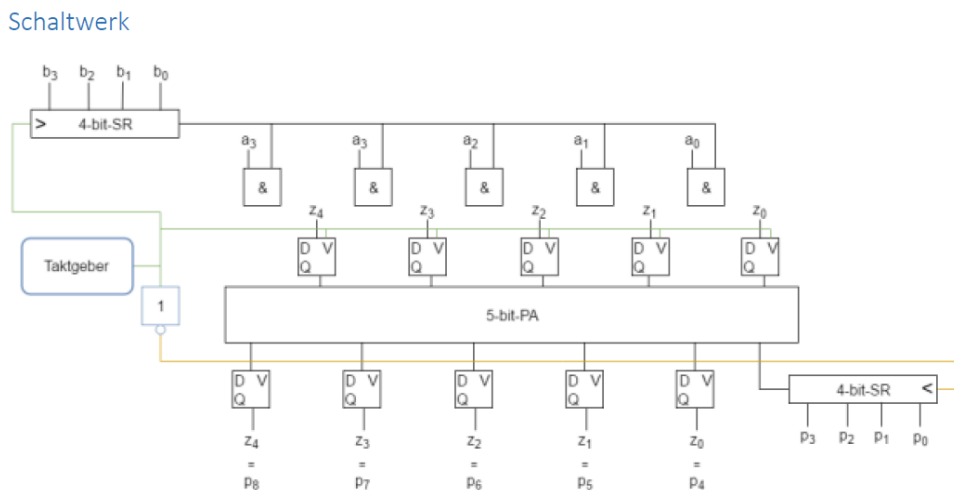


Abbildung 4.11: 5-Bit Seriellmultiplizierer

**Hinweis**

Vor dem ersten Takt müssen alle D-FF auf 0 gesetzt werden („Reset-Eingang“ oder ähnliches)

**Analyse: Hardwareaufwand für  $n \times m$ -Seriellmultiplizierer**

Benötigt werden:

- $n$ -Bit-PA  $\Rightarrow n$ -Bit-RC-PA  $\Rightarrow 18n - 10$  Transistoren
- $n$  UND mit jeweils 2 Eingängen  $\Rightarrow 2$  Transistoren
- $2n$  D-FFs  $\Rightarrow 12n$  Transistoren
- $2m$ -Bit-SR  $\Rightarrow 12m$  Transistoren
- Takterzeugung (wird im folgenden nicht berücksichtigt)

gesamt:  $18n - 10 + 2n + 12n + 12m = 32n + 12m - 10$  Transistoren

$\Rightarrow O(n + m)$

**Analyse: Zeitaufwand für  $n \times m$ -Seriellmultiplizierer mit RC-PA**

Zeitaufwand:  $m$  Taktzyklen

Mindestdauer eines halben Taktzyklus: Berechnungszeit „innen“

$$\frac{1}{2} \text{ Taktzyklus} \geq \text{Auslesezeit(SR)} + \text{UND} + n\text{-Bit-PA} + \text{Einlesezeit(SR)}$$

$$= 1 + 1 + 2n + 1 = 2n + 3 \text{ GLZ}$$

$\Rightarrow 1 \text{ Taktzyklus} \geq 4n + 6 \text{ GLZ}$

gesamt:  $\geq m \cdot (4n + 6) = 4mn + 6m = O(nm)$

Kontrollieren

### 4.9 Division

$$\begin{array}{r}
 47 \ 11 : 13 = 0362, \dots \\
 \underline{0} \\
 47 \\
 \underline{39} \\
 81 \\
 \underline{78} \\
 31 \\
 \underline{26} \\
 5
 \end{array}$$

Die Division wird runter gebrochen auf mehrere Divisionen mit einstelligen Ergebnissen.

Dies ist recht komplex bei einem großem Divisor. Aber im Binärsystem ist es ein einfacher Größenvergleich, da das einstellige Ergebnis nur 1 oder 0 sein kann.

(besser wäre Integration in anschließend notwendiger Subtraktion)

Trotzdem bleiben einige Schritte zu tun, was aufwändig ist. In heutigen CPUs ist deshalb (meist) kein Hardware-Dividierer eingebaut.

*Grund:* Die Division wird deutlich seltener als Addition, Subtraktion und Multiplikation gebraucht.

*Stattdessen:* Division „in Software“/„als Programm“, z. B. als Bibliotheksroutine („Funktion“), evtl. bereitgestellt vom Betriebssystem oder als Mikroprogramm, welches Bestandteil der CPU ist und aufgerufen wird, wenn ein entsprechender Maschinensprachenbefehl ausgeführt werden soll.

### 4.10 CPUs

In Tabelle 4.4 werden RISC und CISC verglichen.

Reduced Instruction Set Computer (RISC)	Complex Instruction Set Computer (CISC)
für jeden Maschinensprachenbefehl gibt es eine „passende“ Hardwareeinheit	manche Maschinensprachenbefehle werden als Mikroprogramm ausgeführt
<ul style="list-style-type: none"> <li>⊕ Einheit für Mikroprogrammausführung kann bei der Produktion entfallen (CPU weniger komplex)</li> <li>⊖ CPU wird komplex, falls die Forderung nach komplexen Befehlen in Maschinensprache realisiert, besteht.</li> <li>⊕ jeder Befehl kann in wenigen (und jeweils einer festen Zahl an) Taktzyklen ausgeführt werden.</li> </ul>	<ul style="list-style-type: none"> <li>⊕ komfortables Programmieren in Maschinensprache</li> <li>⊖ bei manchen Befehlen (welche als Mikroprogramm ausgeführt werden) ist die Bearbeitungszeit sehr groß und variabel.</li> <li>⊖ Mikroprogramme sind Software, komplex und fehleranfällig, ggf. Austausch der CPU für „Bugfix“ notwendig (vgl. Intel Pentium FDIV Bug)</li> </ul>
Einsatz	
PowerPC-CPU in Apple-Rechnern bis vor 10 Jahren oft in „embedded systems“ (Bsp. ARM)	Intel-CPU in heutigen Universalrechnern (und AMD-Äquivalent)

Tabelle 4.4: Vergleich von RISC und CISC

## 4.11 Speicher in Computern

### 4.11.1 Speicherhierarchie

Tabelle 4.5 zeigt die Speicherhierarchie in modernen Universalrechnern

Speicher	von-Neumann-Rechner	Persistenz	Größe	Zugriffszeit
CPU-Register	Zentraleinheit (Rechen- und Steuerwerk)	flüchtig	< 1kB Byte	≈ 200ps
Cache („CPU-Cache“)	nicht vorhanden	flüchtig	~4MB L1≈64KB L2≈512KB	≈ 10ns
Hauptspeicher/ Primärspeicher	Speicherwerk	flüchtig	~8GB	≈ 100ns L1 schneller
Sekundärspeicher	Peripheriegeräte an Ein- und Ausgabe- werk	nicht flüchtig	HDD: 3T SSD: 512GB opt. LW: bis 100GB BLW: wenige TB	HDD ≈ 10ms SSD ≈ 10μs opt. LW ≈ 1s BLW ≈ 1min

Tabelle 4.5: Speicherhierarchie und -Daten

### 4.11.2 Sekundärspeicher

Motivation für Sekundärspeicher: nicht-flüchtig, d. h. der Speicherinhalt bleibt auch bei Stromausfall erhalten.

#### Anwendungen:

- Programmcode (Betriebssystem, Anwendungsprogramme)
- Nutzdaten
- virtuelle Erweiterung des Speicherwerks (Swap-Datei/-Partition)
- Hibernation („Ruhezustand“)

### 4.11.3 CPU-Cache

Temporärer, flüchtiger, schneller Zwischenspeicher, um auf Informationen aus dem Hauptspeicher schneller zugreifen zu können.

Eigenschaften des Cache:

- flüchtig
- kleiner als das zu cachende Medium (Hauptspeicher)
- schneller als das zu cachende Medium (Hauptspeicher)



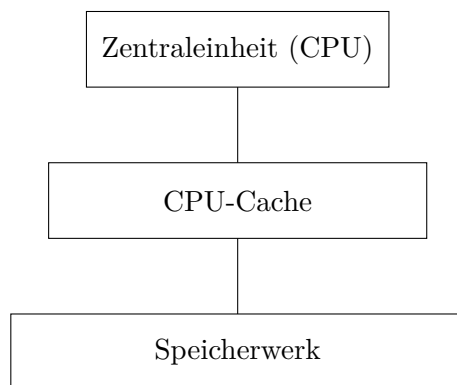


Abbildung 4.12: ...

- transparent, d. h. es wird nicht auf den Cache, sondern auf das zu cachende Medium logisch zugegriffen (die CPU adressiert den Hauptspeicher und nicht den Cache)
- konsistent, d. h. alle Instanzen derselben Hauptspeicheradresse (HSA) haben denselben Wert
- kohärent, d. h. beim Zugriff auf eine HSA wird immer der aktuelle Wert geliefert

**Anmerkung: Kohärenz und Konsistenz**

Kohärenz ist Pflicht und Konsistenz ist Wunsch, da aus Konsistenz Kohärenz folgt. Zeitweise kann aber auf Konsistenz verzichtet werden.

Lokalität des Zugriffsmusters

⇒ verantwortlich dafür, dass der Cache Geschwindigkeitsvorteile bringen kann.

## 5 Abkürzungsverzeichnis

<b>BLW</b>	Bandlaufwerk	
<b>CISC</b>	Complex Instruction Set Computer .....	21
<b>CLA-PA</b>	Carry-Look-Ahead-Paralleladdierer .....	19
<b>CPU</b>	Central Processing Unit	
<b>D-FF</b>	D-Flip-Flop	
<b>DMF</b>	Disjunktive Minimalform .....	8
<b>DNF</b>	Disjunktive Normalform .....	8
<b>ENIAC</b>	Electronic Numerical Integrator and Computer	
<b>IC</b>	Integrated Circuit	
<b>GLZ</b>	Gatterlaufzeit .....	8
<b>HA</b>	Halbaddierer	
<b>HSA</b>	Hauptspeicheradresse .....	23
<b>HW</b>	Hardware	
<b>LW</b>	Laufwerk	
<b>PC</b>	Personal Computer	
<b>PA</b>	Paralleladdierer	
<b>PM</b>	Parallelmultiplizierer .....	18
<b>RC-PA</b>	Ripple-Carry-Paralleladdierer	
<b>RISC</b>	Reduced Instruction Set Computer .....	21
<b>SA</b>	Serielladdierer .....	14
<b>SR</b>	Schieberegister .....	14
<b>UNIVAC</b>	Universal Automatic Computer	
<b>VA</b>	Volladdierer	

## Abbildungsverzeichnis

3.1	Funktionsweise Kathodenstrahlröhre [Quelle: Wikipedia] . . . . .	4
3.2	Vereinfachte Darstellung der von-Neumann-Architektur . . . . .	5
3.3	Vereinfachte Darstellung der Harvard-Architektur . . . . .	6
4.1	Halbaddierer – Schaltnetz und Schaltsymbol . . . . .	9
4.2	Volladdierer – Schaltnetz und Schaltsymbol . . . . .	9
4.3	RC-Paralleladdierer – Schaltnetz und Schaltsymbol . . . . .	10
4.4	Carry-Look-Ahead-Paralleladdierer – Schaltnetz . . . . .	11
4.5	Kaskadierbarer 4-Bit-CLA-PA . . . . .	13
4.6	Serielladdierer . . . . .	14
4.7	Schieberegister und D-Flip-Flop . . . . .	15
4.8	Serielladdierer mit Hardwareoptimierung . . . . .	15
4.9	Schaltnetz Subtrahierer . . . . .	16
4.10	5-Bit Parallelmultiplizierer . . . . .	18
4.11	5-Bit Seriellmultiplizierer . . . . .	20
4.12	... . . . .	23

## Tabellenverzeichnis

3.1	Vergleich der von-Neumann- und Harvard-Architektur . . . . .	7
4.1	Vergleich von Hardwareaufwand eines RC-PA mit dem verbesserten SA . . . . .	16
4.2	Schriftliche Multiplikation „von links nach rechts“ . . . . .	17
4.3	Schriftliche Multiplikation „von rechts nach links“ . . . . .	17
4.4	Vergleich von RISC und CISC . . . . .	21
4.5	Speicherhierarchie und -Daten . . . . .	22

## **Listingsverzeichnis**

## Stichwortverzeichnis

<p>A</p> <p>Addition ..... 8</p> <p>C</p> <p>CISC ..... 21</p> <p>D</p> <p>Division ..... 21</p> <p>DMF ..... 8</p> <p>DNF ..... 8</p> <p>H</p> <p>Halbaddierer ..... 9</p> <p>Harvard-Architektur ..... 5</p> <p>M</p> <p>Multiplikation ..... 17</p> <p>P</p> <p>Paralleladdierer</p>	<p>CLA-PA ..... 11</p> <p>RC-PA ..... 10</p> <p>R</p> <p>RISC ..... 21</p> <p>S</p> <p>Schaltnetz ..... 8</p> <p>Schaltungsanalyse ..... 8</p> <p>Schaltwerk ..... 8</p> <p>Serielladdierer ..... 14</p> <p>Speicher ..... 22</p> <p>Subtraktion ..... 16</p> <p>V</p> <p>Volladdierer ..... 9</p> <p>von-Neumann-Architektur ..... 4</p> <p>Z</p> <p>Zeitaufwand ..... 8</p>
---	---